Efficient monte carlo methods for multi-dimensional learning with classifier chains

Jesse Read [a,*], Luca Martino [a], David Luengo [b]

[a] Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Madrid 28911, Spain
[b] Department of Circuits and Systems Engineering, Universidad Politécnica de Madrid, Madrid 28031, Spain

A B S T R A C T

Multi-dimensional classification (MDC) is the supervised learning problem where an instance is associated with multiple classes, rather than with a single class, as in traditional classification problems. Since these classes are often strongly correlated, modeling the dependencies between them allows MDC methods to improve their performance – at the expense of an increased computational cost. In this paper we focus on the classifier chains (CC) approach for modeling dependencies, one of the most popular and highest-performing methods for multi-label classification (MLC), a particular case of MDC which involves only binary classes (i.e., labels). The original CC algorithm makes a greedy approximation, and is fast but tends to propagate errors along the chain. Here we present novel Monte Carlo schemes, both for finding a good chain sequence and performing efficient inference. Our algorithms remain tractable for high-dimensional data sets and obtain the best predictive performance across several real data sets.

1. Introduction

Multi-dimensional classification (MDC) is the supervised learning problem where an instance may be associated with multiple classes, rather than with a single class as in traditional binary or multi-class single-dimensional classification (SDC) problems. The so-called MDC (e.g., in [1]) is also known in the literature as multi-target, multi-output [2], or multi-objective [3] classification,[1] and is related to multi-task clustering and multi-task learning. The recently popularised task of multi-label classification (see [4–7] for overviews) can be viewed as a particular case of the multi-dimensional problem that only involves binary classes, i.e., *labels* that can be turned on (1) or off (0) for any data instance. The MDC learning context is receiving increased attention in the literature, since it arises naturally in a wide variety of domains, such as image classification [8,9], information retrieval and text categorization [10], automated detection of emotions in music [11] or bioinformatics [10,12].

The main challenge in this area is modeling label dependencies while being able to deal with the scale of real-world problems. A basic approach to MDC is the independent classifiers (IC) method, (commonly known as *binary relevance* in multi-label circles), which decomposes the MDC problem into a set of SDC problems (one per label) and uses a separate classifier for each label variable.[2] In this way, MDC is turned into a series of standard SDC problems that can be solved with any off-the-shelf binary classifier (e.g., a logistic regressor or a support vector machine[3]). Unfortunately, although IC has a low computational cost, it obtains unsatisfactory performance on many data sets and performance measures, because it does not take into account the dependencies between labels [6,14–18].

In order to model dependencies explicitly, several alternative schemes have been proposed, such as the so-called *label powerset* (LP) method [4]. LP considers each potential combination of labels in the MDC problem as a single label. In this way, the multi-dimensional problem is turned into a traditional multi-class SDC problem that can be solved using standard methods. Unfortunately, given the huge number of class values produced by this transformation (especially for non-binary labels), this method is usually unfeasible for practical application, and suffers from issues like overfitting. This was recognised by [14,19], which provide approximations to the LP scheme that reduce these problems, although such methods have been superseded in recent years (as shown in [20]).

* Corresponding author. Tel.: +34 916246005.
E-mail addresses: jesse@tsc.uc3m.es (J. Read), luca@tsc.uc3m.es (L. Martino), david.luengo@upm.es (D. Luengo).

[1] Multi-output, multi-target, multi-variate, etc. can also refer to the regression case, where the outputs are continuous.

[2] Throughout this work we use the term *label* to refer generally to a *class variable* that takes a number of discrete values (i.e., classes); not necessarily binary as in the multi-label case.

[3] Support vector machines (SVMs) are naturally binary, but can be easily adapted to a multi-class scenario by using a pairwise voting scheme, as in [13].

A more recent idea is using classifier chains (CC), which improves the performance of IC and LP on some measures (e.g., the subset 0/1 loss) by constructing a sequence of classifiers that make use of previous outputs of the chain (see [21] for a detailed discussion on MLC methods and loss functions). The original CC method [15] performs a greedy approximation, and is fast (similar to IC in terms of complexity) but is susceptible to error propagation along the chain of classifiers. Nevertheless, a very recent extensive experimental comparison reaffirmed that CC is among the highest-performing methods for MLC, and recommended it as a benchmark algorithm [20].

A CC-based Bayes-optimal method, probabilistic classifier chains (PCC), was recently proposed [16]. However, although it improves the performance of CC, its computational cost is too large for most real-world applications. Some approaches have been proposed to reduce the computational cost of PCC at test time [18,22,23], but the problem is still open. Furthermore, the performance of all CC-based algorithms depends on the label order established at training time, an issue that so far has only been considered by [22] using a heuristic search algorithm called beam search.

In this paper we introduce novel methods that attain the performance of PCC, but remain tractable for high-dimensional data sets both at training and test times. Our approaches are based on a double Monte Carlo optimization technique that, aside from tractable inference, also explicitly searches the space of possible chain-sequences during the training stage. Another advantage of the proposed algorithms is that predictive performance can be traded off for scalability depending on the application. Furthermore, we demonstrate our methods with support vector machine (SVM) as base classifiers (PCC methods have only been used under a logistic regression scheme so far). Finally, unlike the bulk of related literature, we involve the general multi-dimensional scenario (as in [18,2]) and provide a theoretical and empirical analysis of payoff functions for searching the chain space.

A preliminary version of this work has been published in [24]. With respect to that paper, here we introduce three major improvements: we consider the more challenging scenario of multi-dimensional classification (i.e., multi-class labels); at the training stage, we address the problem of finding the optimum label order instead of accepting the original one or using a random label order; for the test stage, we develop a more sophisticated and efficient population Monte Carlo approach for inference.

The paper is organized as follows. In the following Section 2 we review MDC and the important developments leading up to this paper. In Sections 3 and 4 we detail our novel methods for training (including learning the optimum chain sequence) and inference, respectively. In Section 5 we elaborate an empirical evaluation of the proposed algorithms and, finally, in Section 6 we draw some conclusions and mention possible future work.

2. Multi-dimensional classification (MDC)

Let us assume that we have a set of training data composed of N labelled examples, $\mathcal{D}=\{(\mathbf{x}^{(n)},\mathbf{y}^{(n)})\}_{n=1}^{N}$, where

$$\mathbf{x}^{(n)}=[x_1^{(n)},\dots,x_D^{(n)}]^\top \in \boldsymbol{\mathcal{X}}=\mathcal{X}_1\times\cdots\times\mathcal{X}_D\subseteq\mathbb{R}^D$$

is the n-th feature vector (input), and

$$\mathbf{y}^{(n)}=[y_1^{(n)},\dots,y_L^{(n)}]^\top \in \boldsymbol{\mathcal{Y}}=\mathcal{Y}_1\times\cdots\times\mathcal{Y}_L\subset\mathbb{N}_+^L$$

is the n-th label vector (output), with

$$y_\ell^{(n)}\in\mathcal{Y}_\ell=\{1,\dots,K_\ell\},$$

and $K_\ell\in\mathbb{N}_+$ being the finite number of classes associated to the ℓ-th label. The goal of MDC is learning a classification function[4]

$$\mathbf{h}=[h_1,\dots,h_L]^\top : \boldsymbol{\mathcal{X}}\to\boldsymbol{\mathcal{Y}}.$$

Let us assume that the unknown *true* posterior probability density function (PDF) of the data is $p(\mathbf{y}|\mathbf{x})$. From a Bayesian point of view, the optimal label assignment for a given test instance, $\mathbf{x}^*$, is provided by the maximum a posteriori (MAP) label estimate

$$\hat{\mathbf{y}}_{\text{MAP}}=\mathbf{h}_{\text{MAP}}(\mathbf{x}^*)=\operatorname*{argmax}_{\mathbf{y}\in\boldsymbol{\mathcal{Y}}} p(\mathbf{y}|\mathbf{x}^*), \tag{1}$$

where the search must be performed over all possible test labels, $\mathbf{y}\in\boldsymbol{\mathcal{Y}}$. The MAP label estimate is the one most commonly used in the literature, although other approaches are possible, as shown in [16]. Indeed, [16] shows that Eq. (1) minimizes the *exact match* or *subset 0/1* loss, whereas the *Hamming* loss is minimized by finding individual classifiers that maximize the conditional probability for each label. Unfortunately, the problem is further complicated by the fact that the true density, $p(\mathbf{y}|\mathbf{x})$, is usually unknown, and the classifier has to work with an approximation, $\hat{p}(\mathbf{y}|\mathbf{x})$, constructed from the training data. Hence, the (possibly sub-optimal) label prediction is finally given by

$$\hat{\mathbf{y}}=\mathbf{h}(\mathbf{x}^*)=\operatorname*{argmax}_{\mathbf{y}\in\boldsymbol{\mathcal{Y}}} \hat{p}(\mathbf{y}|\mathbf{x}^*). \tag{2}$$

Table 1 summarizes the main notation used throughout this work.

2.1. Multi-dimensional classification vs. multi-label classification

Although binary-only multi-label problems can be considered as a subset of multi-dimensional problems, the reverse is not true, and there are some important quantitative and qualitative differences. Quantitatively, there is a higher dimensionality (for the same value of L); MLC deals with 2^L possible values, whereas MDC deals with $\prod_{\ell=1}^{L}K_\ell$. Note that this affects the inference space, but *not* the sequence space (i.e., the possible orderings of variables). Qualitatively, in MDC the distribution of "labellings" is different, even with binary class variables. In typical MLC problems, the binary classes indicate *relevance* (e.g., the label beach is relevant (or not) to a particular image). Hence, in practice only slightly more than $1/L$ labels are typically relevant to each example on an average [6] (see also Table 5), i.e., $\sum_{\ell=1}^{L}P(y_\ell)\ll L$ where $P(y_\ell)$ is the probability of y_ℓ being relevant. This means that a relatively small part of the $\mathcal{Y}$-space is used. In MDC, classes (including binary classes) are used differently – e.g., a class gender$(\in\{1,2\}\equiv$ M/F) – with a less-skewed distribution of classes; prior-knowledge of the problem aside, we expect $P(Y_\ell=y_\ell)\approx 1/K_\ell$. In summary, in MDC the *practical* $\mathcal{Y}$-space is much greater than in MLC, making probabilistic inference more challenging.

2.2. Independent classifiers (IC)

The method of using *independent classifiers* (IC) on each label is commonly mentioned in the MLC and MDC literature [4,7,15,18]. For each $\ell=1,\dots,L$ a (standard, off-the-shelf binary) classifier h_ℓ is employed to map new data instances to the relevance of the ℓ-th label, i.e.,

$$\hat{\mathbf{y}}=\mathbf{h}(\mathbf{x}^*)=[h_1(\mathbf{x}^*),\dots,h_L(\mathbf{x}^*)]^\top,$$

where, probabilistically speaking, we can define each h_ℓ as

$$\hat{y}_\ell=h_\ell(\mathbf{x}^*)=\operatorname*{argmax}_{y_\ell\in\mathcal{Y}_\ell}\hat{p}(y_\ell|\mathbf{x}^*). \tag{3}$$

[4] We consider $\mathbf{h}$ as a vector because this fits naturally into the independent classifier and classifier chain context, but this is not universal, and $h:\boldsymbol{\mathcal{X}}\to\boldsymbol{\mathcal{Y}}$ is possible in other contexts (such as LP).

Table 1
Summary of the main notation used in this work.

Notation	Description
$\mathbf{x} = [x_1, \ldots, x_D]^\top \in \mathcal{X} \subseteq \mathbb{R}^D$	D-dimensional feature/input vector, with $x_d \in \mathcal{X}_d \subseteq \mathbb{R}$, $d = 1,\ldots,D$
$\mathbf{y} = [y_1, \ldots, y_L]^\top \in \mathcal{Y} \subset \mathbb{N}_+^L$	L-dimensional label/output vector, with $y_\ell \in \mathcal{Y}_\ell = \{1, \ldots, K_\ell\}$ ($K_\ell \geq 2$), $\ell = 1, \ldots, L$
$\boldsymbol{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}] \in \mathcal{X}^N$	$D \times N$ input matrix with all the features
$\boldsymbol{Y} = [\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}] \in \mathcal{Y}^N$	$L \times N$ output matrix with all the labels
$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N = \{\boldsymbol{X}, \boldsymbol{Y}\}$	training data set, $n = 1,\ldots,N$
$p(\mathbf{y}\|\mathbf{x})$	unknown true PDF of the data
$\hat{p}(\mathbf{y}\|\mathbf{x})$	empirical PDF built by the classifier
$\mathbf{x}^* = [x_1^*, \ldots, x_D^*]^\top \in \mathcal{X}$	test feature vector
$\mathbf{h} = [h_1, \ldots, h_L]^\top : \mathcal{X} \to \mathcal{Y}$	classification function built from $\mathcal{D}$
$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [\hat{y}_1, \ldots, \hat{y}_L]^\top$	generic classifier's output
$\hat{\boldsymbol{Y}} = \boldsymbol{H}(\boldsymbol{X}) : \mathcal{X}^N \to \mathcal{Y}^N$	classification matrix $\boldsymbol{H}$ applied to $\boldsymbol{X}$
$\mathbf{s} = [s_1, \ldots, s_L]^\top \in \mathcal{S}^L$	label order, with $s_\ell \in \mathcal{S} = \{1, \ldots, L\}$
$\mathbf{y}_\mathbf{s} = [y_{s_1}, \ldots, y_{s_L}]^\top \in \mathcal{Y}$	L-dimensional permuted label vector
$\mathbf{h}_\mathbf{s} = [h_{s_1}, \ldots, h_{s_L}]^\top : \mathcal{X} \to \mathcal{Y}$	permuted classification function

As we remarked in Section 1, this method is easy to build using off-the-shelf classifiers, but it does not explicitly model label dependencies, and its performance suffers as a result.[5] In fact, it assumes complete independence, i.e., it approximates the density of the data as

$$\hat{p}(\mathbf{y}|\mathbf{x}) = \prod_{\ell=1}^{L} \hat{p}(y_\ell|\mathbf{x}). \tag{4}$$

We always expect label dependencies in a multi-label problem (otherwise we are simply dealing with a collection of unrelated problems); some labels occur more likely together, or mutually exclusively. Thus, it is important to model these dependencies, because doing so can greatly influence the outcome of the predictions.

2.3. Classifier chains (CC)

The *classifier chains* (CC) approach [15] is based on modeling the correlation among labels using the chain rule of probability (see Fig. 1). Given a test instance, $\mathbf{x}^*$, the true label probability may be expressed exactly as

$$p(\mathbf{y}|\mathbf{x}^*) = p(y_1|\mathbf{x}^*) \prod_{\ell=2}^{L} p(y_\ell|\mathbf{x}^*, y_1, \ldots, y_{\ell-1}), \tag{5}$$

Theoretically, label order is irrelevant in Eq. (5), as all the label orderings result in the same PDF. However, since in practice we are modelling an approximation of p (i.e., $\hat{p}$), label order can be very important for attaining a good classification performance, as recognized in [16,21]. Given some label order, $\mathbf{s} = [s_1, \ldots, s_L]^\top$ (a permutation of $\{1, \ldots, L\}$), CC approximates the true data density as

$$\hat{p}(\mathbf{y}_\mathbf{s}|\mathbf{x}^*) = \hat{p}(\mathbf{y}|\mathbf{x}^*, \mathbf{s}) = \hat{p}(y_{s_1}|\mathbf{x}^*) \prod_{\ell=2}^{L} \hat{p}(y_{s_\ell}|\mathbf{x}^*, y_{s_1}, \ldots, y_{s_{\ell-1}}), \tag{6}$$

where $\mathbf{y}_\mathbf{s} = [y_{s_1}, \ldots, y_{s_L}]^\top$ is the permuted label vector (see Fig. 2).

First of all, CC considers an arbitrary label order, $\mathbf{s}$, and learns all the conditional probabilities in (6) from the labelled data during the training stage, thus effectively constructing a chain of classifiers like the one shown in Fig. 1. Then, during the test stage, given a new (test) instance, $\mathbf{x}^*$, CC predicts $\hat{y}_{s_1} = h_{s_1}(\mathbf{x}^*)$ using only the feature vector, whereas for the ℓ-th permuted label ($\ell = 2, \ldots, L$) it also makes use of all the previous predictions $(\hat{y}_{s_1}, \ldots, \hat{y}_{s_{\ell-1}})$, predicting each $\hat{y}_{s_\ell}$ as

$$\hat{y}_{s_\ell} = h_{s_\ell}(\mathbf{x}^*|\hat{y}_{s_1}, \ldots, \hat{y}_{s_{\ell-1}}) = \operatorname*{argmax}_{y_{s_\ell} \in \mathcal{Y}_{s_\ell}} \hat{p}(y_{s_\ell}|\mathbf{x}^*, \hat{y}_{s_1}, \ldots, \hat{y}_{s_{\ell-1}}). \tag{7}$$

Note that, given a data instance $\mathbf{x}^*$ and a label order $\mathbf{s}$, each possible realization of the vector $\mathbf{y}_\mathbf{s}$ can be seen as a path along a tree of depth L, and $\hat{p}(\mathbf{y}_\mathbf{s}|\mathbf{x}^*)$ is the *payoff* or utility corresponding to this path. CC follows a single path of labels $\mathbf{y}_\mathbf{s}$ greedily down the chain of L binary classifiers, as shown in Fig. 3 through a simple example. In carrying out classification down a chain in this way, CC models label dependencies and, as a result, usually performs much better than IC, while being similar in memory and time requirements in practice. However, due to its greedy approach (i.e., only one path is explored) and depending on the choice of $\mathbf{s}$, its performance can be very sensitive to errors, especially in the initial links of the chain [16].

2.4. Probabilistic classifier chains (PCC) and extensions

Probabilistic classifier chains (PCC) was introduced in [16]. In the training phase, PCC is identical to CC; considering a particular order of labels $\mathbf{s}$ (either chosen randomly, or as per default in the dataset). However, during the test stage PCC provides Bayes-optimal inference by exploring all the $\prod_{\ell=1}^{L} K_\ell = 2^L$ possible paths (note that [16] only considers the MLC case, where $K_\ell = 2$ for $\ell = 1, \ldots, L$). Hence, for a given test instance, $\mathbf{x}^*$, PCC provides the optimum $\mathbf{y}_\mathbf{s}$ that minimizes the subset 0/1 loss by maximizing the probability of the complete label vector, rather than the individual labels (as in Eq. (7)), i.e.,

$$\hat{\mathbf{y}}_\mathbf{s} = \mathbf{h}_\mathbf{s}(\mathbf{x}^*) = \operatorname*{argmax}_{\mathbf{y}_\mathbf{s} \in \mathcal{Y}} \hat{p}(\mathbf{y}_\mathbf{s}|\mathbf{x}^*), \tag{8}$$

where $\hat{p}(\mathbf{y}_\mathbf{s}|\mathbf{x}^*)$ is given by (6).[6] In [16] an overall improvement of PCC over CC is reported, but at the expense of a high computational complexity: it is intractable for more than about 10 labels ($\equiv 2^{10}$ paths), which represents the majority of practical problems in the multi-label domain. Moreover, since all the conditional densities in (6) are estimated from the training data, the results can also depend on the chosen label order $\mathbf{s}$, as in CC.

An approximate PCC-based inference method with a reduced computational cost has been proposed in [23]. This approach,

[5] An exception to this rule is the minimization of the Hamming loss, which can be attained by considering each of the individual labels separately. Thus, modeling label dependencies does not provide an advantage in this case, as already discussed in [16,21].

[6] Interestingly, it has been shown in [16,21] that the optimum set of labels that minimize the Hamming loss is given by (3), i.e., the IC approach is optimal for the Hamming loss and no gain is to be expected from any other method that models correlation among labels, with the possible exception of small sample-size problems.

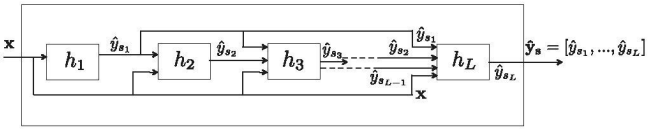

Fig. 1. General scheme of the Classifier Chains (CC) approach.

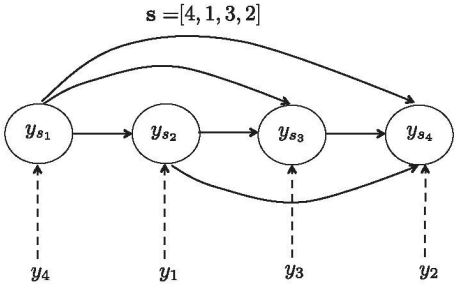

Fig. 2. Example of the permuted label vector in a classifier chain with $L=4$. In this example we have $\mathbf{s} = [4, 1, 3, 2]^\top$, so that $\mathbf{y}_\mathbf{s} = [y_4, y_1, y_3, y_2]^\top$.

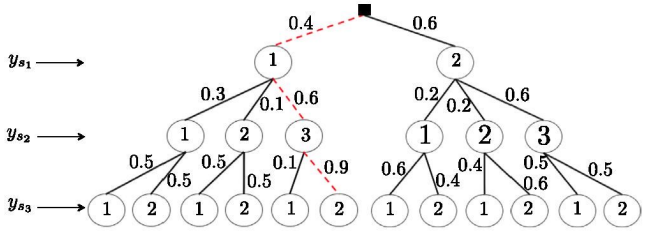

Fig. 3. Example of the $\prod_{\ell=1}^{L} K_\ell = K_1 \times K_2 \times K_3 = 2 \times 3 \times 2 = 12$ possible paths along the tree of class labels y_{s_ℓ} ($\ell = 1, \ldots, L = 3$). The best path, $\mathbf{y}_\mathbf{s} = [1, 3, 2]^\top$, with probability 0.2160, is shown with dashed lines.

named ε-approximate inference, is based on performing a depth-first search in the probabilistic tree with a cutting-off list. It is characterized by quite strong theoretical guarantees regarding the worst-case regret for the subset 0/1 loss and shows a good performance in the experiments, but does not tackle the chain ordering problem. An alternative approach, 'beam search' for PCC, has been proposed in [22]. Beam search is a heuristic search algorithm that speeds up inference considerably and also allows experimentation with chain orderings. Furthermore, the authors of [22] mention the (promising) possibility of using Monte Carlo methods in future works. A simple Monte Carlo-based PCC approach has been considered in [23,25] for maximization of the Hamming and the F-measure loss functions respectively during the test (i.e., inference) stage. We have independently developed a Monte Carlo-based approach in [24], which considers not only the test stage but also the training (i.e., chain order optimization) stage. In this paper we elaborate on this work, providing more sophisticated Monte Carlo algorithms that speed up both the training and test stages.

2.5. Bayesian network classifiers

Conditional dependency networks (CDN) [17] are used as a way of avoiding choosing a specific label order $\mathbf{s}$. Whereas both CC and PCC are dependent on the order of labels appearing in the chain, CDN is a fully connected network comprised of L label-nodes $\hat{p}(y_\ell | \mathbf{x}, y_1, \ldots, y_{\ell-1}, y_{\ell+1}, \ldots, y_L)$ for $\ell = 1, \ldots, L$. Gibbs sampling is used for inference over T steps, and the marginal probabilities collected over the final T_c steps. However, due to having $L(L-1)/2$ links, inference may not scale to large L.

Bayesian classifier chains [18] finds a more tractable (non-fully connected) network based on a maximum spanning tree of label dependencies; although they again use the faster classifier chain-type inference, i.e., by treating the resulting graph as a directed one (by electing one of the nodes to be a root, and thus turning the graph into a tree). This method is similar to CC in the sense that classification depends on the order of nodes, but, unlike CC, it does not model all dependencies (e.g., the dependence between leaf variables is not necessarily modelled).

2.6. Inference in MDC: our approach

As explained in the previous sections, the optimal solution to the classifier chain problem is twofold:

1. Find the best label order $\mathbf{s}$, exploring all the $L!$ possible label orders.
2. Find the best label vector $\mathbf{y}_\mathbf{s}$ within a space composed of $\prod_{\ell=1}^{L} K_\ell$ possible label vectors.

Unfortunately, this task is unfeasible except for very small values of L and K_ℓ ($\ell = 1, \ldots, L$). Indeed, the total space has a cardinality $(\prod_{\ell=1}^{L} K_\ell) \times L!$ (i.e., exponential times factorial). For this reason, in the following we design efficient Monte Carlo techniques to provide good solutions to both problems: finding a good label order $\mathbf{s}$ at the training stage (see Section 3), and then a good label vector $\mathbf{y}$ at the test (i.e., inference) stage (see Section 4).

3. Training stage: finding the best classifier chain

In the training step, we want to learn each of the individual classifiers, h_{s_ℓ} for $\ell = 1, \ldots, L$, and, at the same time, we also wish to find the best chain order, $\mathbf{s} = [s_1, \ldots, s_L]^\top$, out of the $L!$ possibilities. We use a Monte Carlo approach to search this space efficiently.

3.1. Learning the label order

A first, simple, exploration of the label-sequence space is summarized in Algorithm 1. This algorithm can start either with a randomly chosen label order or with the default label order in the dataset, $\mathbf{s}_0$. In each iteration a new candidate sequence $\mathbf{s}'$ is generated randomly according to a chosen proposal density $\pi(\mathbf{s}|\mathbf{s}_{t-1})$ (see Section 3.2 for further details). Then, a suitable payoff function $J(\mathbf{s}')$ is evaluated (see Section 3.3 for a discussion on possible payoff functions). The new candidate label order, $\mathbf{s}'$, is accepted if the value of the payoff function is increased w.r.t. the current one, $\mathbf{s}_{t-1}$ (i.e., if $J(\mathbf{s}') > J(\mathbf{s}_{t-1})$, then $\mathbf{s}_t = \mathbf{s}'$). Otherwise, it is rejected and we set $\mathbf{s}_t = \mathbf{s}_{t-1}$. After a fixed number of iterations T_s, the stored label order $\mathbf{s}_{T_s}$ is returned as the output of the algorithm, i.e., the estimation of the best chain order provided is $\hat{\mathbf{s}} = \mathbf{s}_{T_s}$.[7]

Algorithm 1. Finding a good label order $\hat{\mathbf{s}}$

Input:

- $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{D}$: training data.
- $\pi(\mathbf{s}|\mathbf{s}_{t-1})$: proposal density.
- $\mathbf{s}_0, T_s$: initial label order and number of iterations.

Algorithm:

1. For $t = 1, \ldots, T_s$:
   (a) Draw $\mathbf{s}' \sim \pi(\mathbf{s}|\mathbf{s}_{t-1})$.
   (b) if $J(\mathbf{s}') > J(\mathbf{s}_{t-1})$
   - $\mathbf{s}_t \leftarrow \mathbf{s}'$ accept.

[7] In order to avoid overfitting, this is typically performed using internal train/test split or cross validation, i.e., using part of the training set for building the model and the rest for calculating its payoff. See Section 5 for further details.

(c) else
- $\mathbf{s}_t \leftarrow \mathbf{s}_{t-1}$ reject.

Output:
- $\hat{\mathbf{s}} = \mathbf{s}_{T_s}$: estimated label order.

As we show in Section 3.3, the payoff function $J(\mathbf{s})$ is based on $\hat{p}(\mathbf{y}_\mathbf{s}|\mathbf{x})$, an approximation of the true data density, $p(\mathbf{y}_\mathbf{s}|\mathbf{x})$. Hence, in order to decrease the dependence on the training step we can consider a population of estimated label orders, $\boldsymbol{S} = [\hat{\mathbf{s}}^{(1)}, \ldots, \hat{\mathbf{s}}^{(M)}]$, instead of a single one. This method is detailed in Algorithm 2. The underlying idea is similar to the previous Algorithm 1, but returning the best M label orders (the ones with the highest payoff) after T_s iterations instead of a single label order.

Algorithm 2. Finding a good population of label orders $\boldsymbol{S} = [\hat{\mathbf{s}}^{(1)}, \ldots, \hat{\mathbf{s}}^{(M)}]$

Input:
- $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{D}$: training data.
- $\pi(\mathbf{s}|\mathbf{s}_{t-1})$: proposal density.
- $\mathbf{s}_0, T_s, M$: initial order, number of iterations and population size.

Algorithm:
1. For $t = 1, \ldots, T_s$:
   (a) Draw $\mathbf{s}' \sim \pi(\mathbf{s}|\mathbf{s}_{t-1})$.
   (b) if $J(\mathbf{s}') \geq J(\mathbf{s}_{t-1})$
   - $\mathbf{s}_t \leftarrow \mathbf{s}'$ accept.
   - $w_t \leftarrow J(\mathbf{s}')$ set.

   (c) else
   - $\mathbf{s}_t \leftarrow \mathbf{s}_{t-1}$ accept.
   - $w_t \leftarrow J(\mathbf{s}_{t-1})$ set.
2. Sort $\mathbf{s}_1, \ldots, \mathbf{s}_{T_s}$ decreasingly w.r.t. $w_1, \ldots, w_{T_s}$, taking the top M.

Output:
- $\boldsymbol{S} = [\hat{\mathbf{s}}^{(1)}, \ldots, \hat{\mathbf{s}}^{(M)}]$: population of best M estimated label orders.
- $w^{(1)}, \ldots, w^{(M)}$: corresponding weights.

Once we have described the two proposed Monte Carlo approaches for the training step, the following two sections are devoted to the critical issues for both of the algorithms: the choice of the proposal (Section 3.2) and of the payoff function (Section 3.3).

3.2. Choice of the proposal function

In order to explore the sequence space, $\mathcal{S}$, a proposal mechanism is required. We remark that performing a search in $\mathcal{S}$ requires (**a**) learning a probabilistic model and (**b**) building a new classifier chain for each sequence we want to try. Hence, this stage is inherently much more expensive than searching the label space and the number of label orders that can be explored is thus very limited. Therefore, the proposal density must be both simple and effective. Below, we describe two possibilities.

First proposal scheme: As a first approach we consider a very simple proposal. Specifically, given a sequence

$$\mathbf{s}_{t-1} = [s_{t-1}(1), \ldots, s_{t-1}(L)]^\top,$$

the proposal function $\pi(\mathbf{s}_t|\mathbf{s}_{t-1})$ consists of choosing uniformly two positions of the label order ($1 \leq \ell, m \leq L$) and swapping the labels corresponding to those positions, so that $\mathbf{s}_t(\ell) = \mathbf{s}_{t-1}(m)$ and $\mathbf{s}_t(m) = \mathbf{s}_{t-1}(\ell)$.

Second proposal scheme: The previous proposal does not make a full use of all the available information. For instance, due to the chain structure, changing the initial 'links' in the chain (e.g., $s_t(1)$ or $s_t(2)$) implies a larger jump in the sequence space than changing the final links (e.g., $s_t(L-1)$ or $s_t(L)$). Indeed, if the first L_1 labels in $\mathbf{s}_t$ remain unchanged w.r.t. $\mathbf{s}_{t-1}$, only $L-L_1$ classifiers need to be re-trained, thus saving valuable computation time. In light of this observation, we propose an improvement of the previous proposal based on freezing the links in the chain progressively from the beginning to the end.[8] This allows the algorithm to explore the whole sequence space uniformly in the T_p initial iterations (i.e., potentially requiring re-training of the whole classifier chain), but focuses gradually on the last labels of the sequence, which require almost no re-training and are very cheap to explore. In this case, the first label at the t-th iteration is drawn from

$$p_{\ell,t} \propto \begin{cases} \dfrac{1}{N}, & t \leq T_p; \\ \left(\dfrac{1}{N}\right)^{\beta t/\ell}, & t > T_p; \end{cases} \tag{9}$$

with the second label drawn from

$$p_{m,t} \propto \begin{cases} \dfrac{1}{N-1}, & t \leq T_p; \\ \left(\dfrac{1}{N-1}\right)^{\beta t/m}, & t > T_p, \end{cases} \tag{10}$$

where $\beta > 0$ is a user-defined and constant parameter. First of all, note that the expressions (9) and (10) indicate only the proportionality of the probabilities w.r.t. t and ℓ or m, i.e., in order to obtain the probability mass function we have to normalize the weights above. Moreover, observe that for $t > T_p$ the probability of choosing an index ℓ (resp. m) depends on the position ℓ (resp. m) and the time t. More specifically, this probability increases with the value of ℓ (resp. m), and this effect grows as t increases, with the probability mass function becoming a delta located at the last possible position when $t \to +\infty$. The speed of convergence is controlled by the parameter β: the higher the value of β, the faster Eqs. (9) and (10) become delta functions.

3.3. Cost functions: Bayesian risk minimization

Let us define two matrices, $\boldsymbol{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}]$ and $\boldsymbol{Y} = [\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N)}]$, containing all the features and observations in the training set respectively. Furthermore, let us assume that the data associated with different training instances are independent, i.e.,

$$\hat{p}(\boldsymbol{Y}|\boldsymbol{X}, \mathbf{s}) = \prod_{n=1}^{N} \hat{p}(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathbf{s}) = \prod_{n=1}^{N} \hat{p}(\mathbf{y}_\mathbf{s}^{(n)}|\mathbf{x}^{(n)}). \tag{11}$$

From a Bayesian point of view, the best model (i.e., the best chain or label order) is the one that minimizes the *Bayesian risk* [16,21,27]. Let us define a generic cost function

$$C(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \mathcal{F}(\{\mathcal{L}(\mathbf{y}_\mathbf{s}^{(n)}, \hat{\mathbf{y}}_\mathbf{s}^{(n)})\}_{n=1}^{N}), \tag{12}$$

where we have used $\hat{\boldsymbol{Y}} = \hat{\boldsymbol{Y}}(\mathbf{s}) = \boldsymbol{H}(\boldsymbol{X}|\mathbf{s})$ and $\hat{\mathbf{y}}_\mathbf{s}^{(n)} = \mathbf{h}_\mathbf{s}(\mathbf{x}^{(n)})$ to simplify the notation, $\mathcal{F}(\cdot)$ is a generic functional and $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ is some appropriate loss function, $\mathcal{L}: \mathcal{Y} \mapsto \mathbb{R}$. The Bayesian risk is the expected cost over the joint density of the data given the model

$$\mathcal{R}(\mathbf{s}) = \mathbb{E}_{\mathbf{XY}|\mathbf{s}}\{C(\boldsymbol{Y}, \hat{\boldsymbol{Y}})\}, \tag{13}$$

with $\mathbb{E}_{\mathbf{XY}|\mathbf{s}}$ denoting the mathematical expectation w.r.t. the joint conditional density $\hat{p}(\boldsymbol{X}, \boldsymbol{Y}|\mathbf{s})$, and the optimum chain corresponding

[8] This idea follows the line of the different tempering strategies found in the literature, such as simulated annealing or simulated tempering [26]. However, from a Monte Carlo point of view there is an important difference: our tempering is applied to the proposal, whereas the classical tempering is used to change the target.

to the label order which minimizes this risk. For a given set of training data, the best label order can be determined in a pointwise way by taking the expectation w.r.t. the conditional probability [16,21]:[9]

$$\hat{\mathbf{s}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \mathbb{E}_{\mathbf{Y}|\mathbf{X},\mathbf{s}}\{C(\boldsymbol{Y}, \boldsymbol{H}(\boldsymbol{X}|\mathbf{s}))\}$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{\boldsymbol{Y}\in\mathcal{Y}^N} C(\boldsymbol{Y}, \boldsymbol{H}(\boldsymbol{X}|\mathbf{s})) \prod_{n=1}^{N} \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}), \tag{14}$$

where we have made use of (11) to obtain the last expression.[10]

In the following, we explore several cost and loss functions commonly used in MLC and MDC, showing their probabilistic interpretation from the point of view of finding the best label order.

3.3.1. Additive cost functions

In this section we consider the functional $\mathcal{F}(\cdot) = \sum_{n=1}^{N}(\cdot)$, i.e., an additive cost function. Thus, we have

$$C_{\text{sum}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}). \tag{15}$$

Inserting (15) into (14), and after some algebra, we obtain the following estimator for additive cost functions:

$$\hat{\mathbf{s}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} \sum_{\mathbf{y}_{\mathbf{s}}^{(n)}\in\mathcal{Y}} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}). \tag{16}$$

Unfortunately, minimizing (16) for a generic loss function can be unfeasible in practice. However, by focusing on two of the most common losses used in MLC and MDC (the exact match and the Hamming losses), simple expressions with a straightforward probabilistic interpretation may be found. First of all, let us consider the *exact match* loss,[11] which is defined as

$$\mathcal{L}_{\text{EM}}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) = [\![\, \mathbf{y}_{\mathbf{s}}^{(n)} \neq \hat{\mathbf{y}}_{\mathbf{s}}^{(n)} \,]\!] = \begin{cases} 1, & \mathbf{y}_{\mathbf{s}}^{(n)} \neq \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}; \\ 0, & \mathbf{y}_{\mathbf{s}}^{(n)} = \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}; \end{cases} \tag{17}$$

where $[\![\,\cdot\,]\!]$ returns 1 if its predicate holds and 0 otherwise. Using (17), (16) can be expressed as

$$\hat{\mathbf{s}}_{\text{EM}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} \sum_{\mathbf{y}_{\mathbf{s}}^{(n)}\in\mathcal{Y}} [\![\, \mathbf{y}_{\mathbf{s}}^{(n)} \neq \hat{\mathbf{y}}_{\mathbf{s}}^{(n)} \,]\!] \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)})$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} (1 - \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}))$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmax}} \sum_{n=1}^{N} \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}). \tag{18}$$

From (18) it can be seen that minimizing the exact match loss is equivalent to maximizing the sum of the likelihoods of the predictions for each of the instances in the validation set.[12] Therefore, in order to minimize the exact match loss we should use the following payoff function:

$$J_{\text{EM}}(\mathbf{s}) = \sum_{n=1}^{N} \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}). \tag{19}$$

As a second example, we consider the *Hamming* loss[13]

$$\mathcal{L}_{\text{Ham}}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) = \sum_{\ell=1}^{L} [\![\, y_{s_\ell}^{(n)} \neq \hat{y}_{s_\ell}^{(n)} \,]\!]. \tag{20}$$

Unlike the exact match loss, which returns the same value when $y_{s_\ell}^{(n)} \neq \hat{y}_{s_\ell}^{(n)}$ regardless of how dissimilar they are, the Hamming loss looks at each label component separately. Using (20), it can be shown (see the Appendix) that, for the Hamming loss, (16) becomes

$$\hat{\mathbf{s}}_{\text{Ham}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmax}} \sum_{n=1}^{N} \sum_{\ell=1}^{L} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}). \tag{21}$$

Hence, from (21) we notice that the Hamming loss is minimized by maximizing the sum of the likelihoods of the individual label predictions, given *only* the data, for each of the instances in the validation set.[14] Thus, the corresponding payoff required for minimizing the Hamming loss is

$$J_{\text{Ham}}(\mathbf{s}) = \sum_{n=1}^{N} \sum_{\ell=1}^{L} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}). \tag{22}$$

Note that the CCapproach returns $\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}, \hat{y}_{s_1}^{(n)}, \ldots, \hat{y}_{s_{\ell-1}}^{(n)})$, instead of the PDF required by (21) and (22), $\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)})$. An estimate of $\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)})$ can be obtained by summing over the unnecessary variables

$$\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}) = \sum_{\hat{y}_{s_1}^{(n)}, \ldots, \hat{y}_{s_{\ell-1}}^{(n)}} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}, \hat{y}_{s_1}^{(n)}, \ldots, \hat{y}_{s_{\ell-1}}^{(n)}),$$

but the number of elements in this sum is $\prod_{i=1}^{\ell-1} K_i$, and thus grows exponentially with ℓ (e.g., for MLC $K_i{=}2$ for $i = 1, \ldots, \ell$, so $\prod_{i=1}^{\ell-1} K_i = 2^{\ell-1}$). Hence, a better alternative is computing $\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)})$ directly during the training stage, as done by the IC approach, instead of the $\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}, \hat{y}_{s_1}^{(n)}, \ldots, \hat{y}_{s_{\ell-1}}^{(n)})$ required by the CC approach.

3.3.2. Multiplicative cost functions

As a second family of cost functions we consider multiplicative cost functions, i.e., we consider a functional $\mathcal{F}(\cdot) = \prod_{n=1}^{N}(\cdot)$, which leads us to

$$C_{\text{prod}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \prod_{n=1}^{N} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}). \tag{23}$$

Inserting (23) into (14), the estimator is now given by

$$\hat{\mathbf{s}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{\boldsymbol{Y}\in\mathcal{Y}^N} \prod_{n=1}^{N} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)})$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \prod_{n=1}^{N} \sum_{\mathbf{y}_{\mathbf{s}}^{(n)}\in\mathcal{Y}} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)})$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} \log\left(\sum_{\mathbf{y}_{\mathbf{s}}^{(n)}\in\mathcal{Y}} \mathcal{L}(\mathbf{y}_{\mathbf{s}}^{(n)}, \hat{\mathbf{y}}_{\mathbf{s}}^{(n)}) \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}) \right), \tag{24}$$

which has a similar functional form to (16), with the log of the inner sum inside the outer sum. Hence, following an identical procedure to the one in Eq. (18) for the exact match loss, we obtain

$$\hat{\mathbf{s}}_{\text{EM}-\text{prod}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} \log(1 - \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}))$$
$$= \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmax}} \prod_{n=1}^{N} \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}), \tag{25}$$

[9] Note that in [16,21] this approach is followed to find the best classifier for a given label order, whereas here we use it to find the best label order (i.e., the best model).

[10] In practice, we use internal validation to avoid overfitting, i.e., the training set is divided into two: a first part for training the classifiers and a second part for validation. Thus, all the expressions in this section should really consider only the validation set, which will be a subset of the training set. However, in the following we always consider $n = 1, \ldots, N$ for the sake of simplicity.

[11] Also called by some authors the *subset*0/1 loss (cf. [16]).

[12] Note that this is equivalent to the result obtained in [16,21] for the test stage, i.e., for inferring the best $\hat{\mathbf{y}}_{\mathbf{s}}$ for a given label order $\mathbf{s}$.

[13] The name is due to the fact that it corresponds to the Hamming distance for the binary labels used in MLC. Although this is no longer true for the non-binary labels that can appear in MDC, this definition is still valid and we keep the name used in MLC.

[14] Once more this is equivalent to the result obtained in [16,21] for the test stage.

which corresponds to the maximum of the likelihood function. Hence, the corresponding payoff function is precisely the likelihood function:

$$J_{\mathrm{EM-prod}}(\mathbf{s}) = \prod_{n=1}^{N} \hat{p}(\hat{\mathbf{y}}_{\mathbf{s}}^{(n)}|\mathbf{x}^{(n)}), \tag{26}$$

Similarly, following the steps shown in the Appendix for the additive cost function, we may obtain the estimator for the Hamming loss in the multiplicative case:

$$\hat{\mathbf{s}}_{\mathrm{Ham-prod}}(\boldsymbol{X}) = \underset{\mathbf{s}\in\mathcal{S}^L}{\operatorname{argmax}} \prod_{n=1}^{N}\sum_{\ell=1}^{L} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}). \tag{27}$$

which is similar to (25), but now the product is on the individual label likelihoods instead of the global likelihoods of the different instances. The payoff function in this case is

$$J_{\mathrm{Ham-prod}}(\mathbf{s}) = \prod_{n=1}^{N}\sum_{\ell=1}^{L} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}). \tag{28}$$

4. Test (inference) stage: finding the best label vector

In the test stage, for a given test instance $\mathbf{x}^*$ and a label order $\mathbf{s}$, our aim is finding the optimal label vector $\hat{\mathbf{y}}_{\mathbf{s}}$ that maximizes Eq. (8). The PCC method [16] solves this part analytically (by performing an exhaustive search). However, since this method becomes computationally intractable for anything but small L (the full space involves $\prod_{\ell=1}^{L} K_\ell$ possible paths), the goal is providing a Monte Carlo (MC) approximation of the estimated label vector

$$\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} \approx \hat{\mathbf{y}}_{\mathbf{s}} = \underset{\mathbf{y}_{\mathbf{s}}}{\operatorname{argmax}}\, \hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x}^*) \tag{29}$$

for the minimization of the exact-match loss or

$$\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} \approx \hat{\mathbf{y}}_{\mathbf{s}} = \underset{\mathbf{y}_{\mathbf{s}}}{\operatorname{argmax}} \sum_{\ell=1}^{L} \hat{p}(y_{s_\ell}|\mathbf{x}^*) \tag{30}$$

for the Hamming loss, such that $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} \to \hat{\mathbf{y}}_{\mathbf{s}}$ when $T_y \to +\infty$, with T_y being the number of iterations of the MC algorithm.

A first possible MC approach for the minimization of the exact match loss is provided by Algorithm 3.[15,16] Given a test instance $\mathbf{x}^*$ and a label order $\mathbf{s}$, this algorithm starts from an initial label vector $\mathbf{y}_{\mathbf{s}}^{(0)}$ arbitrarily chosen (e.g., randomly or from the greedy inference offered by standard CC), and draws samples $\mathbf{y}_{\mathbf{s}}^{(i)}$ $(i=1,\ldots,T_y)$ directly from the model learnt in the training stage, $\hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x}^*)$.[17] Then, the label vector $\mathbf{y}_{\mathbf{s}}^{(k)}$ with the highest payoff is returned as the output, i.e., $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} = \mathbf{y}_{\mathbf{s}}^{(k)}$, with

$$k = \underset{i=1,\ldots,T_y}{\operatorname{argmax}}\, \hat{p}(\mathbf{y}_{\mathbf{s}}^{(i)}|\mathbf{x}^*) \tag{31}$$

for the minimization of the exact-match loss and

$$k = \underset{i=1,\ldots,T_y}{\operatorname{argmax}} \sum_{\ell=1}^{L} \hat{p}(y_{s_\ell}^{(i)}|\mathbf{x}^*) \tag{32}$$

when the goal is minimizing the Hamming loss. From a Monte Carlo point of view,it is important to remark that all the candidate vectors $\mathbf{y}'$ are always drawn *directly* from the target density, $\hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x}^*)$,i.e., $\mathbf{y}'$ is always a valid path on a tree selected according to the weights of the different branches. This is an important consideration,since it guarantees that the estimated label vector, $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})}$,will always be a feasible path.

Algorithm 3. Obtaining $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} \approx \hat{\mathbf{y}}_{\mathbf{s}}$ that minimizes the exact-match loss for a given test instance $\mathbf{x}^*$.

Input:
- $\mathbf{x}^*, \mathbf{s}$: test instance and given label order.
- $\hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x})$: probabilistic model.
- $\mathbf{y}_{\mathbf{s}}^{(0)}, T_y$: initial label vector and number of iterations.

Algorithm:
1. For $t = 1,\ldots,T_y$:
   (a) Draw $\mathbf{y}_{\mathbf{s}}^{'} \sim \hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x}^*)$.
   (b) `if` $\hat{p}(\mathbf{y}_{\mathbf{s}}^{'}|\mathbf{x}^*) > \hat{p}(\mathbf{y}_{\mathbf{s}}^{(t-1)}|\mathbf{x}^*)$
   - $\mathbf{y}_{\mathbf{s}}^{(t)} \leftarrow \mathbf{y}_{\mathbf{s}}^{'}$ accept.

   (c) `else`
   - $\mathbf{y}_{\mathbf{s}}^{(t)} \leftarrow \mathbf{y}_{\mathbf{s}}^{(t-1)}$ reject.

Output:
- $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} = \mathbf{y}_{\mathbf{s}}^{(T_y)}$: predicted label assignment.

As previously discussed, the inference of Algorithm 3 depends strictly on the chosen label order $\mathbf{s}$. For this reason, we also propose another scheme that uses a population of label orders $\boldsymbol{S} = [\mathbf{s}^{(1)},\ldots,\mathbf{s}^{(M)}]$ (chosen randomly or obtained using Algorithm 2). A naive procedure to incorporate this information in the inference technique would be running M parallel algorithms to find sequences of labels $\mathbf{y}_{\mathbf{s}}^{(i)}$ (like Algorithm 3) using different label orders $\mathbf{s}^{(i)}$ $(i=1,\ldots,M)$ and then selecting the best one. However, this approach is computationally inefficient. In Algorithm 4 we propose a more sophisticated approach that makes use of the information within the entire population $\boldsymbol{S}$ but requires running only one random search. The main steps of the method can be summarized as follows:

1. A label order $\mathbf{s}' \in \boldsymbol{S}$ is selected according to some weights (e.g., those provided by Algorithm 2) proportional to a certain payoff function.
2. A good label vector $\hat{\mathbf{y}}$ is found by following Algorithm 3.
3. The procedure is repeated T_y times, with the best label vector for the T_s label orders explored being returned as the output.

Algorithm 4. Obtaining $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} \approx \hat{\mathbf{y}}_{\mathbf{s}}$ that minimizes the exact-match loss given $\mathbf{x}^*$,and a population $\boldsymbol{S}$.

Input:
- $\mathbf{x}^*$: test instance.
- $\boldsymbol{S} = [\mathbf{s}^{(1)},\ldots,\mathbf{s}^{(M)}]$: population of M label orders.
- $w^{(1)},\ldots,w^{(M)}$: corresponding weights.
- $\hat{p}(\mathbf{y}_{\mathbf{s}}|\mathbf{x}) = \hat{p}(\mathbf{y}|\mathbf{x},\mathbf{s})$: probabilistic model.
- T_s, T_y: number of iterations for searching $\mathbf{s}$ and $\mathbf{y}_{\mathbf{s}}$ resp.
- $\mathbf{y}_{\mathbf{s}}^{(0)}$: initial label vector.

Algorithm:
1. For $t_1 = 1,\ldots,T_s$:
   (a) Choose $\mathbf{s}_{t_1} = s^{(j)} \sim w^{(j)}/\sum_{i=1}^{M} w^{(i)}$ for $j=1,\ldots,M$.
   (b) Set $\mathbf{z}_1 = \mathbf{y}_{\mathbf{s}_{t_1-1}}^{(t_1-1)}$.
   (c) For $t_2 = 1,\ldots,T_y$:
      i. $\mathbf{z}' \sim \hat{p}(\mathbf{z}|\mathbf{x}^*,\mathbf{s}_{t_1})$.

[15] Algorithm 3 can also be used to minimize the Hamming loss, simply changing the condition in step 1(b) by the following condition:

$$\sum_{\ell=1}^{L} \hat{p}(y_{s_\ell}^{'}|\mathbf{x}^*) > \sum_{\ell=1}^{L} \hat{p}(y_{s_\ell}^{(t-1)}|\mathbf{x}^*),$$

where $\hat{p}(y_{s_\ell}^{(t-1)}|\mathbf{x}^*)$ can be computed as described at the end of Section 3.3.1

[16] An MC-based approach like the one shown in Algorithm 3 has been independently proposed in [23] for the minimization of the exact match loss during the test stage.

[17] Note that this is equivalent to generating random paths in the tree of class labels according to the corresponding weights associated to each branch (see Fig. 3).

ii. if $\hat{p}(\mathbf{z}'|\mathbf{x}^*, \mathbf{s}_{t_1}) > \hat{p}(\mathbf{z}_{t_2}|\mathbf{x}^*, \mathbf{s}_{t_1})$
- $\mathbf{z}_{t_2+1} \leftarrow \mathbf{z}'$ accept.

iii. else
- $\mathbf{z}_{t_2+1} \leftarrow \mathbf{z}_{t_2}$ reject.

(d) Set $\mathbf{y}_{\mathbf{s}_{t_1}}^{(t_1)} = \mathbf{z}_{T_y}$.

Output:
- $\hat{\mathbf{y}}_{\mathbf{s}}^{(\mathbf{mc})} = \mathbf{y}_{\mathbf{s}_{T_s}}^{(T_s)}$: predicted label assignment.

5. Experiments

In order to compare fairly both the performance and the computational effort, we progressively apply the ideas introduced in the previous sections to form four novel methods:

- MCC (Algorithm 3): given a classifier chain trained on some previously determined label order $\mathbf{s}$ (e.g., randomly as in CC or PCC), we infer the label vector for all test instances using a simple MC approach.
- $\mathrm{M_s CC}$ (Algorithm 1 plus Algorithm 3): like MCC, but we additionally search for a suitable label order $\hat{\mathbf{s}}$ during the training stage. Specifically, we use Algorithm 1 with the simplest proposal density $\pi(\mathbf{s}|\mathbf{s}_{t-1})$ described in the first part of Section 3.2.
- $\mathrm{PM_s CC}$ (Algorithm 2 plus Algorithm 4): population version of $\mathrm{M_s CC}$, still using the simplest proposal density $\pi(\mathbf{s}|\mathbf{s}_{t-1})$ described in Section 3.2.
- $\mathrm{P_t M_s CC}$ (Algorithm 2 plus Algorithm 4): $\mathrm{PM_s CC}$ with the improved proposal $\pi(\mathbf{s}|\mathbf{s}_{t-1})$ described in the last part of Section 3.2.

Note that MCC and $\mathrm{M_s CC}$ differ on how they obtain the label order $\mathbf{s}$ (randomly chosen for MCC or estimated using Algorithm 1 for $\mathrm{M_s CC}$), whereas $\mathrm{PM_s CC}$ and $\mathrm{P_t M_s CC}$ differ on the proposal used to search for the best label order.

5.1. Comparison of different cost functions

In this section we analyze the performance of different payoff functions: the two additive payoffs given by (19) and (22), and the multiplicative payoff of Eq. (26). Initially we focus on the Music dataset, because it is faster to run and easier to visualise than other datasets. Indeed, since $L=6$ (see Table 5) we can find the optimum label order for the exact match payoff ($\hat{\mathbf{s}}_{\mathrm{EM}} = [3,5,0,1,4,2]^\top$) by performing an exhaustive search over the $L! = 720$ possibilities. Table 2 shows that the proposed Monte Carlo approach (Algorithm 1) arrives to the optimum label order under two separate initializations after 1935 and 1626 iterations; although we note that after a much smaller number of iterations (310 and 225 respectively), the difference is minimal in payoff. The search also converges maximizing $J_{\mathrm{EM-prod}}$ (Table not displayed), although we noted that it is a *different* maxima, specifically, $\hat{\mathbf{s}}_{\mathrm{EM-prod}} = [4,5,1,2,3,0]^\top$.

A similar analysis may be performed for other datasets where the optimum label order cannot be found by exhaustive search. Fig. 4 plots similar statistics for the Yeast data ($L=14$), whereas Fig. 5 shows the payoffs when maximizing them *separately* (using the same random seed in both graphs) for Enron ($L=53$). All these analyses suggest that the payoff functions are climbing the same terrain, but there are many peaks of similar height. Thus, while $\hat{\mathbf{s}} = [4,5,1,2,3,0]^\top$ does not appear superficially close to $\hat{\mathbf{s}} = [3,5,0,1,4,2]^\top$, as found by using the different payoff functions on Music, both result in higher performance than selecting $\mathbf{s}$ randomly. This is also confirmed by the results of predictive performance shown later on, and justifies searching the $\mathcal{S}$-space. The fact that many label orders provide good results, as opposed to just one, is not unexpected and justifies our population Monte Carlo method (Algorithm 2). As a general remark, we also note that the terrain of $J_{\mathrm{EM-prod}}$ appears much rougher: when maximizing $J_{\mathrm{EM-prod}}$, J_{EM}'s appreciation is still relatively smooth, but not vice versa.

Table 2
Running algorithm 1 on Music dataset using payoff J_{EM} under $T_s = \infty$ (i.e., run until convergence to the optimum label order, $\hat{\mathbf{s}}_{\mathrm{EM}} = [3,5,0,1,4,2]^\top$, obtained through an exhaustive search). We only show the iterations where a new $\mathbf{s}_t \leftarrow \mathbf{s}'$ is accepted (plus the default $\mathbf{s}_0$); displaying also the payoffs J_{EM}, $J_{\mathrm{EM-prod}}$ (in the log domain) and J_{Ham} (note that these numbers have *not* been normalized by N). The experiment is performed twice for two different random seeds (i.e., starting from a different $\mathbf{s}_0$).

t	$\mathbf{s}_t^\top$	J_{EM}	$J_{\mathrm{EM-prod}}$	J_{Ham}
(a) Random seed 1				
0	[5, 2, 4, 1, 0, 3]	164.92	−1079.26	2889.89
2	[5, 4, 0, 1, 2, 3]	166.14	−1084.91	2887.3
4	[5, 3, 0, 2, 1, 4]	166.89	−1085.4	2886.84
310	[5, 3, 0, 1, 4, 2]	167.42	−1084.58	2887.14
492	[3, 5, 1, 0, 4, 2]	167.53	−1083.97	2887.41
1682	[3, 0, 5, 1, 4, 2]	167.62	−1082.89	2887.94
1935	[3, 5, 0, 1, 4, 2]	167.73	−1082.79	2887.91
Random seed 2				
0	[4, 2, 0, 1, 3, 5]	155.7	−1109.58	2864.41
1	[4, 2, 0, 3, 1, 5]	156.87	−1102.82	2868.08
2	[4, 2, 0, 3, 5, 1]	159.45	−1096.79	2873.95
3	[4, 0, 2, 3, 5, 1]	161.79	−1091.6	2880.18
5	[4, 0, 5, 2, 3, 1]	163.14	−1093.32	2880.16
18	[5, 1, 4, 3, 2, 0]	163.59	−1085.69	2885.91
23	[5, 4, 0, 1, 2, 3]	166.12	−1084.97	2887.24
128	[3, 5, 1, 0, 2, 4]	167.05	−1084.7	2887.2
176	[5, 3, 1, 0, 4, 2]	167.22	−1085.75	2886.65
225	[5, 3, 1, 4, 0, 2]	167.41	−1083.1	2887.93
1422	[3, 5, 1, 4, 0, 2]	167.69	−1081.35	2888.68
1626	[3, 5, 0, 1, 4, 2]	167.73	−1082.82	2887.9

Table 3 compares the predictive performance using different payoff functions. J_{EM} performs better than $J_{\mathrm{EM-prod}}$, which has a rougher terrain to climb. However, we remark again that all of them are better than choosing a random $\mathbf{s}$. This corresponds with our intuition and theoretical results, although perhaps even more experimentation is necessary to resolve the question in the formal statistical significant sense; due to randomly varying the $\mathcal{D}$-split and the initial $\mathbf{s}_0$, the payoff function, plus taking into account the huge $\mathcal{S}$ space, a vast number of experiments would be necessary for getting conclusive statistical-significant figures. We instead decided to invest more computation in our large-scale comparison between methods (Section 5.3).

5.2. Comparison of MCC with other MLC approaches

Table 4 outlines all the methods used in the experiments, their parameters, and relevant references. We compare to baseline IC, the original classifier chains method CC, the ensemble version ECC, the Bayes-optimal rendition PCC and the ε-approximate and beam search variants; the conditional dependency networks method CDN; and RF-PCT, a decision-tree based approach.

As a base classifier (for all methods relying on one) we mainly use support vector machines (SVMs) fitted with logistic models (as according to [13]) so as to provide a probabilistic output and otherwise with the default parameters as in the SMO implementation of the Weka framework [28]. Logistic regression has so far been a popular choice in the probabilistic multi-label literature (e.g., [16,17]) due to its probabilistic output. However, we have found that SVM-based methods can perform better, at least without tuning the parameters. For best accuracy, it is highly recommended to tune the base classifier. However, we wish to avoid this "dimension" and instead focus on the multi-label methods.

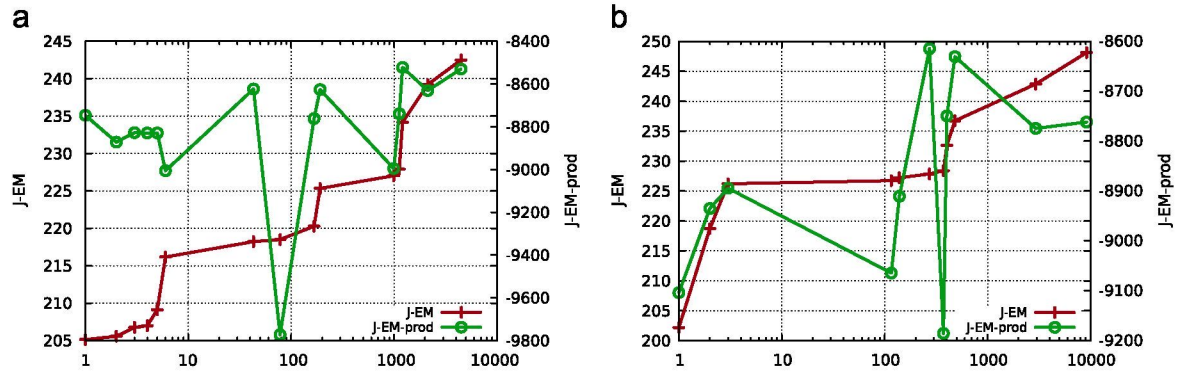

Fig. 4. A plotted version of Table 2 for the Yeast data, for both J_{EM} and $J_{\text{EM}-\text{prod}}$ up till $T_s = 10,000$. The left vertical axis corresponds to J_{EM}, and the right vertical axis to $J_{\text{EM}-\text{prod}}$. Note the log-scale horizontal axis. (a) Random seed 1 and (b) Random seed 2.

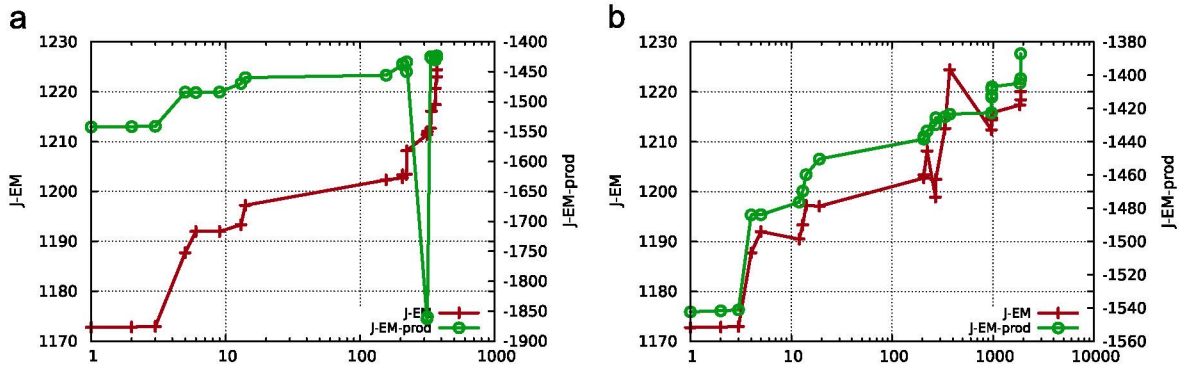

Fig. 5. Searching **s** space on the Enron dataset, displaying *both* payoff scores, but maximizing only *one* of them (J_{EM} left, $J_{\text{EM}-\text{prod}}$ right) – using the same random seed (initial $\mathbf{s}_0$); until $T_s = 10,000$. The left vertical axis corresponds to J_{EM}, and the right vertical axis to $J_{\text{EM}-\text{prod}}$. Note the log-scale horizontal axis. (a) Maximizing $J_{\text{EM}-\text{prod}}$ and (b) Maximizing J_{EM}.

Table 3
Average exact match and Hamming score across 100 experiments (2/3:1/5 random data split), under different payoff functions, including *none* (i.e., random **s**).

Payoff	Exact match	Payoff	Ham. Score
(a) Music, $T_s = 1500$			
J_{EM}	0.3386 ± 0.026	J_{EM}	0.7998 ± 0.011
$J_{\text{EM}-\text{prod}}$	0.3307 ± 0.025	$J_{\text{EM}-\text{prod}}$	0.7982 ± 0.010
J_{Ham}	0.3319 ± 0.024	J_{Ham}	0.7988 ± 0.010
None	0.3264 ± 0.022	None	0.7935 ± 0.014
(b) Yeast, $T_s = 100$			
J_{EM}	0.2107 ± 0.0126	J_{EM}	0.7815 ± 0.0047
$J_{\text{EM}-\text{prod}}$	0.2071 ± 0.0108	$J_{\text{EM}-\text{prod}}$	0.7816 ± 0.0043
J_{Ham}	0.2104 ± 0.0115	J_{Ham}	0.7804 ± 0.0046
None	0.2055 ± 0.0115	None	0.7802 ± 0.0048

All our methods are implemented and will be made available within the Meka framework[18]; an open-source framework based on the Weka machine learning framework [28] with added support for multi-label classification and evaluation. Table 5 displays the collection of real world datasets that we use; most are familiar to the MLC and MDC literature [14–16].

The two contrasting measures EXACT MATCH LOSS (Eq. (17)) and HAMMING LOSS (Eq. (20)) are almost invariably used in the multi-label literature, so we use them here. Note that in some results we pose both as a payoff/score $1 - \mathcal{L}$, where $\mathcal{L}$ denotes normalized loss; in other words, EXACT MATCH and HAMMING SCORE, where 1.0 is the best possible performance.

5.3. Results

Table 7 displays the average results of 5-fold cross validation. Results for running time performance are given in Table 8. All experiments were carried out on Intel Xeon CPUs at 3.16 GHz allowing up to 2 GB of RAM. The ranks and average ranks of each method are displayed, and significance according to the Nemenyi test [29]; where *amac*; *sc*; *b* indicates that algorithm *a* is significantly better than *b* (under a *p*-value of 0.10).

In Table 6, to compare with existing methods from the literature, we have taken results from [23,22,20], and displayed results for our methods alongside using the same train/test splits.[19] Note that the PCC methods use logistic regression as a base classifier, and RF-PCT is decision-tree based.

As in the literature, CC improves over IC considerably, particularly under EXACT MATCH, where label dependence must be modelled for best performance. PCC improves further on CC – in the cases where it is tractable – also across both evaluation measures.

In Table 7 we see that MCC outperforms CC on almost every occasion (only two exceptions) and is identical to PCC on all datasets where PCC completes, indicating that our methods conduct accurate inference like PCC, but are much more computationally tractable. Recall that PCC's inference is *optimal*.

Overall, M_SCC obtains similar performance to MCC. It would follow that higher performance could be obtained with a higher value of T_s, hinted at by the fact that most wins over M_SCC are on the smaller datasets (SolFlare, Bridges, Music) where the chain-sequence space is smaller and easier to explore (with small T_s). Of

[18] http://meka.sourceforge.net

[19] As made available on the Mulan website: http://mulan.sourceforge.net/datasets.html

Table 4
The methods considered and their parameters. The novel methods proposed are below the middle line; where each inherits the parameters of the previous ones, e.g., $\text{P}_t\text{M}_\mathbf{s}\text{CC}$ takes parameters $T_y = 100, T_s = 50, M = 10, \beta = 0.03$. For CDN, T_c is the number of collection iterations. M indicates generally the number of models. We selected the 'width' parameter configuration for the Beam Search that gave the best 0/1 Loss rank in [22], and similarly for ε-approximate PCC in [23]. RF-PCT parameters are as in [20].

Key	Method	Parameters	Reference
IC	Independent classifiers		[4]
CC	Classifier chains		[15]
ECC	Ensembles of classifier chains	$M=10$	[15]
PCC	Probabilistic classifier chains		[16]
CDN	Conditional dependency networks	$T=1000$ $T_c=100$	[17]
PCC-ε	ε-approx. PCC	$\varepsilon=0.25$	[23]
PCC-Beam	Beam search PCC	$w_s^y = \frac{5}{1}$	[22]
RF-PCT	Random forest of PCTs	$M=100$	[2]
MCC	Monte Carlo optimization for CC	$T_y=100$	Algorithm 1
$\text{M}_\mathbf{s}\text{CC}$	$2 \times$ MCC	$T_s=50$	Algorithms 1 and 3
$\text{PM}_\mathbf{s}\text{CC}$	Population $\text{M}_\mathbf{s}\text{CC}$	$M=10$	Algorithms 2 and 4
$\text{P}_t\text{M}_\mathbf{s}\text{CC}$	Population $\text{M}_\mathbf{s}\text{CC}$ (w/ temperature)	$\beta=0.03$	Algorithms 2 and 4

Table 5
A collection of datasets and associated statistics, where LC is *label cardinality*: the average number of labels relevant to each example; relevant for binary labels [4]. We have divided multi-dimensional datasets, and multi-label (binary-only) datasets.

Dataset	N	L	K	d	LC	Type
Solar Flare	323	3	5	10	N/A	Astrology
Bridges	107	5	2–6	7	N/A	Civil engineering
Thyroid	9172	7	2–5	28	N/A	Medical
Parkinson's	488	5	3	58	N/A	Medical
Music	593	6	2	72	1.87	Audio
Scene	2407	6	2	294	1.07	Image
Yeast	2417	14	2	103	4.24	Biology
Genbase	661	27	2	1185	1.25	Biology
Medical	978	45	2	1449	1.25	Medical/text
Enron	1702	53	2	1001	3.38	Email/text
Reuters	6000	103	2	500	1.46	News/text
TMC2007	28596	22	2	500	2.16	Text
MediaMill	43907	101	2	120	4.38	Video

Table 6
Comparison of other methods from the literature on the train/test splits used in these papers. N/A indicates that the result is not available (the dataset was not used by the algorithm's authors). DNF indicates did Not Finish in 24 h. Ranks are not shown due to many missing values. Best results are highlighted in **bold**.

Dataset	PCC-ε	PCC-Beam	RF-PCT	MCC	$\text{M}_\mathbf{s}\text{CC}$	$\text{P}_t\text{M}_\mathbf{s}\text{CC}$
(a) 0/1 Exact-Match Loss						
Music	0.718	0.673	0.693	0.673	0.688	**0.653**
Scene	0.385	0.362	0.482	0.419	0.382	**0.360**
Yeast	0.764	**0.758**	0.848	0.775	0.776	0.776
Genbase	N/A	**0.020**	N/A	**0.020**	**0.020**	**0.020**
Medical	0.541	**0.360**	0.372	0.364	**0.360**	**0.360**
Enron	0.848	**0.805**	0.869	0.822	0.870	0.867
TMC2007	**0.718**	N/A	0.816	0.796	DNF	DNF
MediaMill	N/A	N/A	**0.878**	0.913	DNF	DNF
(b) Hamming Loss						
Music	0.219	0.221	**0.189**	0.226	0.216	0.202
Scene	0.107	0.106	**0.094**	0.119	0.110	0.105
Yeast	0.211	0.210	**0.197**	0.211	0.213	0.209
Genbase	N/A	**0.001**	N/A	**0.001**	**0.001**	**0.001**
Medical	0.015	**0.011**	0.014	**0.011**	**0.011**	**0.011**
Enron	0.046	0.052	0.046	0.053	0.58	0.58
TMC2007	0.055	N/A	**0.011**	0.076	DNF	DNF
MediaMill	N/A	N/A	**0.029**	0.034	DNF	DNF

course, increasing T_s implies a correspondingly increased computational cost. On the other hand, it is likely that this issue stems from the fact that a single chain sequence may not necessarily be best for predicting *all* test instances. This was a motivation behind our population-of-**s** method, $\text{PM}_\mathbf{s}\text{CC}$.

In Table 7 $\text{PM}_\mathbf{s}\text{CC}$ obtains the best performance of all methods, under both Exact Match and Hamming Loss, across the majority of datasets. It appears that on Medical and Enron the random chains of ECC provide better performance. Perhaps adding an ECC-like voting scheme would make up the difference. $\text{P}_t\text{M}_\mathbf{s}\text{CC}$ obtains almost as good performance as $\text{PM}_\mathbf{s}\text{CC}$, but is more efficient on most datasets. This is exactly what it was designed to be: an efficient version of $\text{PM}_\mathbf{s}\text{CC}$. However, we were surprised to see in Table 8 that it is not more efficient on some of the large datasets, indicating that there may be some overhead in our implementation which is sensitive to L. In Table 6 competition to $\text{P}_t\text{M}_\mathbf{s}\text{CC}$ is shown by PCC-Beam under Exact Match, and RF-PCT under Hamming Loss. It is worth noting that the Beam-search implementation paid more attention to setting up the base classifier, and we select the ${}_5{}^1$ configuration from [22] as the best of several combinations, whereas we used a single combination of $T_y/T_s/M$ parameters in our MCC methods. Both chaining approaches are quite competitive, especially taking into account that RF-PCT was pegged one of the best of 12 methods in [20]'s evaluation. It is true that PF-PCT is more efficient. The PCC-Beam paper does not report results for larger datasets like TMC2007 and MediaMill.

Although in Table 6 the methods which search the label sequence space do not finish within the 24 h cut off, we point out that for $\text{P}_t\text{M}_\mathbf{s}\text{CC}$ this is a less-than linear increase with T_s (depending on β). With $T_s = 10$, we expect the method to take < 10 times longer than MCC. Further speedups are possible, for example using faster classifiers and/or smaller sets for internal validation. We intend to investigate this in future work. Nevertheless, it is clear that searching the chain space in this fashion becomes less feasible for larger amounts of data.

Our methods are generally faster than CDN, especially for larger L. This is interesting, since an attraction of '*chain-less*' methods like CDN is that no study of chain sequence is necessary. However, we see that in this case, although there is no need to choose a chain-sequence, inference is relatively much more costly.

As a side note, we point out that there is clearly a qualitative difference between the multi-dimensional datasets where $K_\ell > 2$, and the binary-labelled datasets, where $K_\ell = 2$. For example, on Bridges the otherwise-more-advanced methods (ECC and MCC and variations thereof) perform relatively poorly compared to the basic baseline IC, which performs best. More than anything this is probably due to the relatively smaller size of these datasets,

Table 7
Predictive Performance from 5-fold CV, displayed as: value rank, i.e., the average value across all folds and the rank of that value for each dataset. Note that the rank is based on a higher precision than shown in the table.

Dataset	IC	CC	PCC	ECC	CDN	MCC	M_sCC	PM$_s$CC	P$_t$M$_s$CC
(a) EXACT MATCH									
SolFlare	0.77 7	0.80 1	0.78 5	0.69 8	0.59 9	0.78 5	0.79 2	0.79 2	0.79 2
Bridges	0.09 9	0.12 4	0.12 4	0.10 8	0.14 1	0.12 4	0.13 2	0.13 2	0.11 7
Parkins	0.17 1	0.17 2	0.16 5	0.16 8	0.16 5	0.16 5	0.16 8	0.17 4	0.17 2
Thyroid	0.83 6	0.02 9	0.84 3	0.82 7	0.78 8	0.84 3	0.84 3	0.84 2	0.85 1
Music	0.30 7	0.29 9	0.35 4	0.31 6	0.30 8	0.35 4	0.36 3	0.37 1	0.37 2
Scene	0.54 8	0.55 7	0.64 3	0.61 6	0.53 9	0.64 3	0.63 5	0.68 2	0.69 1
Yeast	0.14 7	0.15 6	DNF	0.19 5	0.07 8	0.21 4	0.22 3	0.23 1	0.22 2
Genbase	0.94 8	0.96 2	DNF	0.94 6	0.94 6	0.96 2	0.96 2	0.96 5	0.96 1
Medical	0.58 8	0.62 4	DNF	0.64 1	0.60 7	0.63 2	0.62 3	0.62 5	0.60 6
Enron	0.07 8	0.10 3	DNF	0.11 1	0.07 7	0.10 2	0.09 6	0.09 5	0.10 4
Reuters	0.29 7	0.35 6	DNF	0.36 5	0.27 8	0.37 4	0.37 1	0.37 1	0.37 3
avg. rank	6.91	4.82	4.00	5.55	6.91	3.45	3.45	2.73	2.82
(b) HAMMING SCORE									
SolFlare	0.90 7	0.92 1	0.90 4	0.85 8	0.77 9	0.90 4	0.90 6	0.90 2	0.90 2
Bridges	0.63 7	0.66 3	0.67 1	0.64 6	0.62 9	0.67 1	0.65 4	0.65 4	0.63 8
Parkins	0.68 2	0.68 1	0.67 7	0.68 3	0.68 3	0.67 7	0.67 7	0.68 3	0.68 6
Thyroid	0.97 1	0.83 9	0.97 1	0.97 7	0.96 8	0.97 1	0.97 6	0.97 1	0.97 1
Music	0.81 4	0.79 8	0.80 6	0.81 5	0.79 9	0.80 6	0.81 3	0.81 1	0.81 1
Scene	0.89 7	0.86 8	0.89 4	0.90 3	0.86 9	0.89 4	0.89 6	0.90 2	0.91 1
Yeast	0.79 1	0.75 7	DNF	0.79 3	0.72 8	0.78 6	0.79 4	0.79 2	0.79 4
Genbase	1.00 7	1.00 1	DNF	1.00 7	1.00 5	1.00 1	1.00 1	1.00 5	1.00 1
Medical	0.99 5	0.99 2	DNF	0.99 1	0.99 8	0.99 2	0.99 2	0.99 5	0.99 5
Enron	0.93 2	0.92 5	DNF	0.94 1	0.92 3	0.92 5	0.92 8	0.92 5	0.92 3
Reuters	0.98 1	0.98 1	DNF	0.98 1	0.98 8	0.98 1	0.98 1	0.98 1	0.98 1
avg. rank	4.00	4.18	3.83	4.09	7.18	3.45	4.36	2.82	3.00

Nemenyi signif.: MCC*mac*; *sc*; IC; MCC*mac*; *sc*; CDN; M_sCC*mac*; *sc*; IC; M_sCC*mac*; *sc*; CDN; PM$_s$CC*mac*; *sc*; IC; PM$_s$CC*mac*; *sc*; CDN; P$_t$M$_s$CC*mac*; *sc*; IC; P$_t$M$_s$CC*mac*; *sc*; CDN.
Nemenyi signif.: PCC*mac*; *sc*; CDN; MCC*mac*; *sc*; CDN; PM$_s$CC*mac*; *sc*; CDN; P$_t$M$_s$CC*mac*; *sc*; CDN.

Table 8
Running time; averaged over 5-fold CV and rounded nearest second. Dataset-wise rankings, and some of the smaller datasets are not shown due to space limitations.

Dataset	IC	CC	PCC	ECC	CDN	MCC	M_sCC	PM$_s$CC	P$_t$M$_s$CC
(a) RUNNING TIME (training+testing)									
Scene	12	11	15	44	92	90	1347	684	335
Yeast	11	11	DNF	66	88	149	1313	731	546
Genbase	11	8	DNF	56	573	1695	5287	774	823
Medical	9	11	DNF	86	1546	3420	6940	1038	1192
Enron	102	92	DNF	349	3091	3884	10821	2986	3470
Reuters	106	120	DNF	20593	14735	1837	5740	4890	5310
(b) BUILD TIME (training only)									
Scene	12	10	11	43	13	13	1233	671	322
Yeast	11	11	DNF	64	12	15	1164	707	525
Genbase	9	7	DNF	46	5	11	3875	651	683
Medical	8	8	DNF	63	7	12	4986	835	961
Enron	99	86	DNF	307	72	80	8474	2729	3139
Reuters	96	102	DNF	2030	120	108	4920	4120	4449

making it difficult to get a good approximation $\hat{p}(\mathbf{y}|\mathbf{x}) \approx p(\mathbf{y}|\mathbf{x})$ of the true density. In future work we intend to create larger MDC ($K_\ell > 2$) datasets and investigate this more thoroughly.

6. Conclusions and future work

We designed novel Monte Carlo (MC) schemes for inference in a multi-dimensional learning framework using classifier chains. MC techniques are used to efficiently search the chain-order space at the training stage, and the label-path space at the inference stage. We analysed several possible choices of payoff functions for these MC methods, both from a practical and theoretical point of view. An extensive empirical evaluation showed that our techniques yield better predictive performance than many related methods while remaining computationally tractable. Our model convincingly obtains overall the best predictive performance of all the methods we looked at, and proves tractable enough for many real-world applications. The MC approach is interesting for, and applicable, to a wide range of problems, and can be scaled up using faster base classifiers (either in building the final model and/or the internal models during the chain-space search). In future work, we also intend to look at more advanced search algorithms and dependency structures other than chain models.

Conflict of interest

None declared.

Acknowledgements

This work has been partly supported by the Spanish government through projects COMONSENS (CSD2008-00010), ALCIT (TEC2012-38800-C03-01), DISSECT (TEC2012-38058-C03-01) and COMPREHENSION (TEC2012-38883-C02-01). We would also like to thank the associate editor and the reviewers for their helpful comments and corrections.

Appendix A. Additive cost function for the Hamming Loss

Inserting (20) into (16), we notice that the optimum label order for the additive cost function with the Hamming loss is given by

$$\hat{\mathbf{s}}_{\text{Ham}}(\boldsymbol{X}) = \underset{\mathbf{s} \in \mathcal{S}^L}{\operatorname{argmin}} \sum_{n=1}^{N} \sum_{\mathbf{y}_{\mathbf{s}}^{(n)} \in \mathcal{Y}} \sum_{\ell=1}^{L} [\![y_{s_\ell}^{(n)} \neq \hat{y}_{s_\ell}^{(n)}]\!] \hat{p}(\mathbf{y}_{\mathbf{s}}^{(n)} | \mathbf{x}^{(n)})$$

$$= \operatorname*{argmin}_{\mathbf{s}\in\mathcal{S}^L} \sum_{n=1}^{N}\sum_{\ell=1}^{L}\sum_{y_{s_\ell}^{(n)}\in\mathcal{Y}_{s_\ell}} [\![y_{s_\ell}^{(n)} \neq \hat{y}_{s_\ell}^{(n)}]\!] \hat{p}(y_{s_\ell}^{(n)}|\mathbf{x}^{(n)})$$

$$\times \sum_{\mathbf{y}_{\mathbf{s}\setminus s_\ell}^{(n)}\in\mathcal{Y}\setminus\mathcal{Y}_{s_\ell}} \prod_{\substack{k=1\\k\neq\ell}}^{L} \hat{p}(y_{s_k}^{(n)} \,|\, \mathbf{x}^{(n)}, y_{s_\ell}^{(n)}, y_{s_1}^{(n)}, \ldots, y_{s_{k-1}}^{(n)}), \tag{A.1}$$

where we have used the chain rule of probability with $y_{s_\ell}^{(n)}$ as root node, $\mathbf{y}_{\mathbf{s}\setminus s_\ell}^{(n)} = [y_{s_1}^{(n)}, \ldots, y_{s_{\ell-1}}^{(n)}, y_{s_{\ell+1}}^{(n)}, \ldots, y_{s_L}^{(n)}]^\top$, and the last expression has been obtained simply separating the ℓ-th label from the rest. Now, noticing that the last term in (A.1) is equal to one, we obtain Eq. (21):

$$\hat{\mathbf{s}}_{\mathrm{Ham}}(\boldsymbol{X}) = \operatorname*{argmin}_{\mathbf{s}\in\mathcal{S}^L} \sum_{n=1}^{N}\sum_{\ell=1}^{L}\sum_{y_{s_\ell}^{(n)}\in\mathcal{Y}_{s_\ell}} [\![y_{s_\ell}^{(n)} \neq \hat{y}_{s_\ell}^{(n)}]\!] \hat{p}(y_{s_\ell}^{(n)}|\mathbf{x}^{(n)})$$

$$= \operatorname*{argmin}_{\mathbf{s}\in\mathcal{S}^L} \sum_{n=1}^{N}\sum_{\ell=1}^{L} (1-\hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)})) = \operatorname*{argmax}_{\mathbf{s}\in\mathcal{S}^L} \sum_{n=1}^{N}\sum_{\ell=1}^{L} \hat{p}(\hat{y}_{s_\ell}^{(n)}|\mathbf{x}^{(n)}). \tag{A.2}$$

References

[1] C. Bielza, G. Li, P. Larrañaga, Multi-dimensional classification with Bayesian networks, International Journal of Approximate Reasoning 52 (2011) 705–727.

[2] D. Kocev, C. Vens, J. Struyf, S. Deroski, Tree ensembles for predicting structured outputs, Pattern Recognition 46 (2013) 817–833.

[3] D. Kocev, C. Vens, J. Struyf, S. Džeroski, Ensembles of multi-objective decision trees, in: Proceedings of the 18th European Conference on Machine Learning, ECML '07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 624–631.

[4] G. Tsoumakas, I. Katakis, Multi label classification: an overview, International Journal of Data Warehousing and Mining 3 (2007) 1–13.

[5] A.C. Carvalho, A.A. Freitas, A tutorial on multi-label classification techniques, in: A. Abraham, A.-E. Hassanien, V. Snášel (Eds.), Foundations of Computational Intelligence Volume 5, Studies in Computational Intelligence, vol. 205, Springer, 2009, pp. 177–195.

[6] J. Read, Scalable Multi-label classification (Ph.D. thesis), University of Waikato, 2010.

[7] G. Tsoumakas, I. Katakis, I. Vlahavas, Mining multi-label data, in: O. Maimon, L. Rokach (Eds.), Data Mining and Knowledge Discovery Handbook, 2nd edition,Springer, 2010.

[8] M.R. Boutell, J. Luo, X. Shen, C.M. Brown, Learning multi-label scene classification, Pattern Recognition 37 (2004) 1757–1771.

[9] G.-J. Qi, X.-S. Hua, Y. Rui, J. Tang, H.-J. Zhang, Two-dimensional multilabel active learning with an efficient online adaptation model for image classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (2009) 1880–1897.

[10] M.-L. Zhang, Z.-H. Zhou, Multilabel neural networks with applications to functional genomics and text categorization, IEEE Transactions on Knowledge and Data Engineering 18 (2006) 1338–1351.

[11] K. Trohidis, G. Tsoumakas, G. Kalliris, I. Vlahavas, Multilabel classification of music into emotions, in: ISMIR '08: 9th International Conference on Music Information Retrieval.

[12] Z. Barutcuoglu, R.E. Schapire, O.G. Troyanskaya, Hierarchical multi-label prediction of gene function, Bioinformatics 22 (2006) 830–836.

[13] T. Hastie, R. Tibshirani, Classification by pairwise coupling, in: M.I. Jordan, M. J. Kearns, S.A. Solla (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 10, MIT Press, 1998.

[14] G. Tsoumakas, I.P. Vlahavas, Random k-labelsets: an ensemble method for multilabel classification, in: ECML '07: 18th European Conference on Machine Learning, Springer, 2007, pp. 406–417.

[15] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine Learning 85 (2011) 333–359.

[16] W. Cheng, K. Dembczyński, E. Hüllermeier, Bayes optimal multilabel classification via probabilistic classifier chains, in: 27th International Conference on Machine Learning (ICML), Haifa, Israel.

[17] Y. Guo, S. Gu, Multi-label classification using conditional dependency networks, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1300–1305.

[18] J.H. Zaragoza, L.E. Sucar, E.F. Morales, C. Bielza, P. Larrañaga, Bayesian chain classifiers for multidimensional classification, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI).

[19] J. Read, B. Pfahringer, G. Holmes, Multi-label classification using ensembles of pruned sets, in: ICDM'08: Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 995–1000.

[20] G. Madjarov, D. Kocev, D. Gjorgjevikj, S. Deroski, An extensive experimental comparison of methods for multi-label learning, Pattern Recognition 45 (2012) 3084–3104.

[21] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, On label dependence and loss minimization in multi-label classification, Machine Learning 88 (2012) 5–45.

[22] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Learning and inference in probabilistic classifier chains with beam search, in: Machine Learning and Knowledge Discovery in Databases, vol. 7523, pp. 665–680.

[23] K. Dembczyński, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification, in: Workshop Proceedings of 20th European Conference on Artificial Intelligence (ECAI), Montpellier, France, pp. 294–299.

[24] J. Read, L. Martino, D. Luengo, In: Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing, ICASSP '13, IEEE, 2012; pp. 3457–3461.

[25] K.J. Dembczynski, W. Waegeman, W. Cheng, E. Hüllermeier, An exact algorithm for F-measure maximization, in: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 24, 2011, pp. 1404–1412.

[26] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimisation by simulated annealing, Science 220 (1983) 671–680.

[27] H.L.V. Trees, Detection, Estimation, and Modulation Theory. Part I: Detection, Estimation, and Linear Modulation Theory, John Wiley & Sons, New York, NY, USA, 2001.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, R. Peter, I.H. Witten, The WEKA data mining software: an update, SIGKDD Explorations 11 (2009) 10–18.

[29] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.